Some notes on getting the JFE model to converge.

The model and algorithm parameters are set in a file that I call GetEq\_blah.c where 'blah' describes the model's parameters. For example, GetEq\_0v0\_j2b6.c implies the model has traders with '0' private value being the only informed traders (i.e., v0 for me indicates a zero lag between common value changes and observed changes) and j2 means the common value jumps 2 ticks at a time and b6 means traders observe details in the book corresponding to when nBT=6.

This parameter file loads the main code via its last line: #include "GetEq\_state-action\_code.c" Note that GetEq\_state-action\_code.c in turn loads invcdfn.c These three files comprise all the code for the model.

Each "run" (i.e., market specification) has its own directory of output files (which serve as input files when restarting or simulating the model). This directory is specified by #define TMP. The parameters have two main "configurations": Obtaining Equilibrium and Simulating Trades.

A number of parameters must be set in a consistent manner for each of these two modes.

1. Obtaining Equilbrium (in which value functions are updated)

Recommended / Required param values:

* nJ 1e7 (initially at 1e7, then 3e8 for final convergence)
* mJ 0
* UPDATE\_s0 1
* UPDATE\_sync 0
* UPDATE\_maxPVT 1.0 (.1 if only updating beliefs of deviators in information acquisition stage which precedes the trading game)
* PrTremble .02 (initially set high (say .05), then lower to .01 or .005 for final convergence. Probably do last Nreset when start at lowest PrTremble )
* READ 1 ( 1 to load state space from ODD nJ iteration. 2 to load from EVEN iteration. 0 if fresh start )
* WRITE 1 ( will save state space each nJ trades )

If the state-space is getting too large, the user can use the PURGE option. To purge states that were not even checked (let alone chosen), after the READ from files, set

* PURGE\_MODE -1
* PURGE\_NC 1
* PURGE\_N 0

2. Simulating Trade Data (holding value functions fixed). Recommended / Required param values:

* nJ 1e6
* mJ 1e6
* UPDATE\_s0 0
* UPDATE\_sync 0
* UPDATE\_maxPVT 0.0 (.1 if only updating beliefs of deviators in information acquisition stage which precedes the trading game)
* PrTremble .0 ( NO trembles when simulating actual data. Trembles only used to obtain equilibrium)
* READ 1 ( 1 to load state space from ODD nJ iteration. 2 to load from EVEN iteration. 0 if fresh start )
* WRITE 0

Within the mode for obtaining equilibrium, the user will probably need stages:

1. A first stage in which READ 0 and nJ is low (say 1e6 or 1e7) which enables the model to quickly get to the equilibrium "ballpark".
2. A second stage in which READ is 1 or 2 and the user restarts the code with nJ high (at least 1e8).

In the initial convergence stage, the denominator (i.e., s->N ) should be reset to be low again every once in a while since the "reinforcement learning" is faster when the denominator is low since the value function is simply an average of realized payoffs each time a state-action pair is taken. However, complete convergence relies on the law of large numbers, so s->N should NOT be lowered in the latter stage of convergence. The lowering of s->N is controlled by the Nreset parameters.

I usually use the following scheme regarding Nreset:

1. initially set Nreset 2 (when nJ is low). This resets s->N to be no higher than 2\*(jj+1) each nJ trader choices, where jj is the counter for number of times nJ traders have been processed. Hence, as the algorithm progresses, the resetting of N is less severe.
2. after killing the job and restarting with higher nJ (say 3e8), I set Nreset -1 and Nreset\_xNN 10 which is a "proportional resetting" in which each set of nJ trader actions will receive a weight of no less than 1/10. The idea here is that we want learning to be relatively fast (meaning new realizations will get reasonably high weights and weights on the old realizations which were from out-of-equilibrium play get lower weight).
3. when really near convergence I set Nreset -1 and Nreset\_xNN 99 (or even higher), or simply set Nreset 0 so that s->N is never lowered.

The above Nreset strategy implies manully killing the first two runs and recompiling/restarting for the next runs. I usually run the first mode for 10-30 minutes, then mode 2 for 3-5 hours. Then do a PURGE if needed (to get rid of states that were only encountered when the model was far from equilibrium) and re-run for another 3-5 hours. I run in mode 3 until the number of new states encountered is small (say .1% of nJ) and the Chi-sq is small (below 2).

To get a model converged, I usually have to re-start the code 2 or 3 times with different settings on some of the algorithm parameters. For example, I usually start with higher trembling rates, and then take PrTremble down to 0.01 or so. Note that when I run the model to get a final stream of simulated T&Q data I turn off the trembles (PrTremble = 0). That is, trembles are only used to compute the equilibria. We don't conjecture that traders indeed make mistakes.

The true Chi-sq convergence test, which far exceeds ad hoc convergence tests used by other simulation methods, is very hard to satisfy FOR LARGE NUMBERS of nJ. Indeed, for any given tolerance of value functions being close to equilibrium, there exists some nJ large enough that the Chi-sq test will fail. In my experiences, the simulated outcomes of interest seem to settle down well before the chi-sq using nJ of 1e8 falls below 2. The true Chi-sq test should be done in simulation mode (when value functions are fixed). The chi-sq test stat will be higher when UPDATE\_s0 is 0 than 1 (since UPDATE\_s0 0 compares ultimate payoffs to expected payoffs when the trader FIRST entered the market).

Starting from scratch with a new model, you need to specify the expected payoff for non-orders (i.e., doing nothing until the trader returns again). This is given by the matrix CS\_non . If the first element of this matrix is < -9 then this matrix is read in from a file created during the most recent nJ simulated traders. But, the first time it is run, this file won't exist yet. The matrices values shouldn't matter much. They should reflect your best guess of the Consumer Surplus (i.e., payoff) of each trader type. This payoff is then discounted by the expected time until the trader has another chance to take an action.